# INF-253 Lenguajes de Programación

## Tarea 5: Prolog

02 de noviembre de 2023

## 1. Detective Sholmes

El joven Herlock Sholmes, aspirante a ser un famoso y renombrado detective, postula a la Academia de Investigaciones y Operaciones Especiales de Dronles justo cuando cumple 16 años. La academia solo acepta a los mejores y más astutos estudiantes en sus aulas, haciendo que pasen por una prueba de admisión, que pone en juego las habilidades lógicas, creativas y conocimiento de programación de sus postulantes. Habiendo inscrito su postulación, Herlock se devuelve contento a su hogar, aunque un poco dudoso sobre cómo será la prueba, puesto que solo le dijeron cuándo sería, pero no dónde ni cómo será, antes de irse le comentaron que "espere en su casa, descansado y preparado", así que regresa listo para hacer justo eso.

En esta tarea se presentarán distintos problemas a resolver con el lenguaje Prolog, el cual se puede encontrar en: https://www.swi-prolog.org/download/stable.

## 2. Descifrar el mensaje

En el día de la prueba, llega a su correo una carta con una nota adentro: *"1100100011001000"* y en el reverso *"Bases ADN"*. La nota es bien confusa, esperable viniendo de la academia, pero relacionando los mensajes logra suponer que las bases del ADN deben ser la clave para descifrar el código.

Si divide el mensaje en secciones de 2 bits, existen combinaciones suficientes en cada sección para representar las 4 bases (ignorando Uracilo), así, puede crear un predicado que transforme el código en una serie de bases. Aquí se presentan los hechos para hacer equivalentes los bits con las bases:

```
cifrado([0,0], a).

cifrado([0,1], g).

cifrado([1,0], c).

cifrado([1,1], t).
```

Se relacionan 2 bits del mensaje con 1 base, resultando muy fácil descifrarlo. Por ejemplo, insertar "*00110110"* entrega como respuesta *"atgc"*. En particular, el formato de los mensajes será de la siguiente forma:

Ejemplos:

```
?- descifrar([0,0,1,1,0,1,1,0], R).

R = [a, t, g, c].

?- descifrar([0,0,1], R).

false.
```

Su tarea es implementar la regla *descifrar(mensaje, Respuesta)* que se capaz de instanciar en *Respuesta* la traducción del mensaje.

**Nota:** El último caso del ejemplo da cuenta de que el programa no necesita ser capaz de resolver mensajes con cantidad impar de cifras.

## 3. La contraseña de la caja fuerte

Justo después de descifrar el mensaje, la academia ha enviado una caja fuerte cerrada -*¿Cómo saben que ya fue descifrado el mensaje?*- con la siguiente nota dentro: "Cree un predicado que le ayude a abrir esta caja fuerte".

Considerando la regla *cerradura($x_1$, $x_2$, $x_3$, $x_4$, $x_5$)*, definida en la primera línea de su código con valores reales (como *cerradura(1,2,3,4,5)*), debe crear un predicado en Prolog que reciba una combinación ingresada por el usuario, y responder sobre qué tan cerca está el detective de descifrar la contraseña correcta. "Cerca" y "Lejos" serán las respuestas del programa, e indican qué tanto se está acercando el usuario a descubrir la contraseña. La respuesta se determina a partir de la distancia entre la suposición del usuario y la contraseña real, a continuación, se refleja la ecuación que determina qué es "Cerca":

$$E = \frac{\Delta x_1 + \Delta x_2 + \Delta x_3 + \Delta x_4 + \Delta x_5}{5} < 1$$

Con $\Delta x_n$ siendo la **diferencia absoluta** entre la n-ésima cifra de la cerradura y la n-ésima cifra de la suposición del usuario. El caso opuesto, donde $E \geq 1$ indica que el supuesto se encuentra "Lejos".

Por ejemplo, si al inicio de su código tiene *cerradura(1, 4 ,5 ,1, 0)*, y el usuario consulta la contraseña *1 0 0 0 0*, el pr debe responder que la solución real se encuentra "Lejos" de la que acaba de ingresar.

$$E = \frac{|1-1| + |4-0| + |5-0| + |1-0| + |0-0|}{5} = 2 \rightarrow E > 1$$

Ejemplos:

```
?- verificar(1, 0, 0, 0, 0, R).
R = “Lejos”.
?- verificar(1, 4, 5, 1, 1, R).
R = “Cerca”.
?- verificar(1, 4, 5, 1, 0, R).
R = “Contraseña descubierta”.
```

Su tarea es implementar la regla *verificar(X1, X2, X3, X4, X5, R)*, que recibe 5 números y retorna en *R* la instanciación de la respuesta.

## 4. Mejora para las pistas

Con lo anterior listo, el postulante se percató que en verdad la pista de cerca o lejos no es muy buena para llegar a la contraseña de la cerradura. Por ello, decidió realizar un predicado diferente, que escuche los sonidos que hace la caja cuando ingresa las cifras y responda con cuántos aciertos hubo en la prueba.

Ahora, cuando el usuario intente verificar la combinación, el programa responderá con un número que indica cuántas de las cifras coinciden con la contraseña de la cerradura. Es decir, si la regla en el código es *cerradura(1, 4, 5, 1, 0)* y el usuario prueba verificar la combinación *1 0 0 0 0*, la respuesta debe ser 2, pues el primer y el último digito de cerradura coinciden con la combinación propuesta.

Ejemplos:

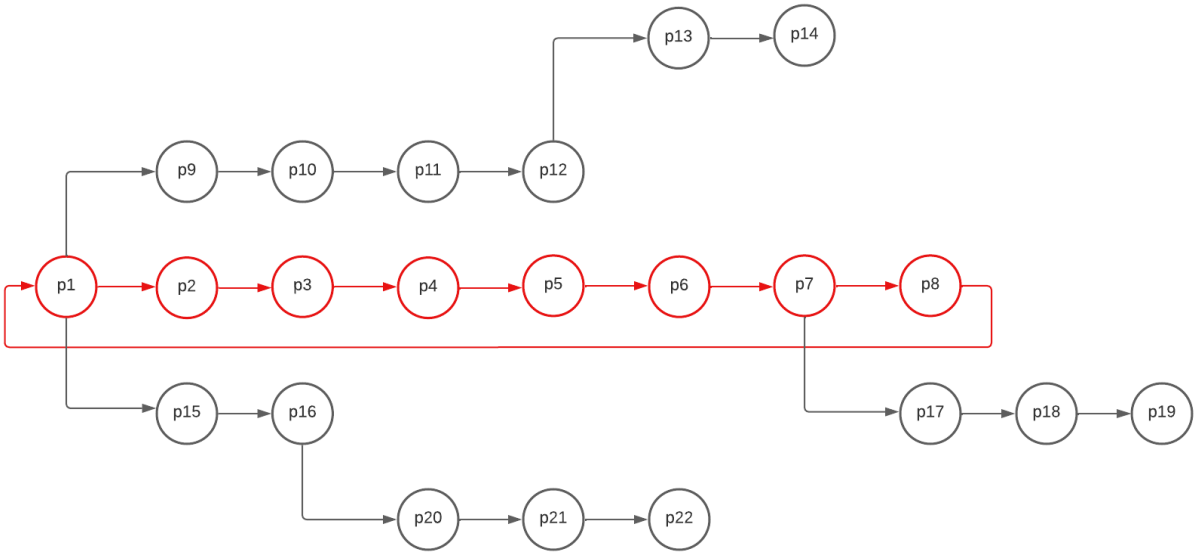
```
?- verificar(1, 0, 0, 0, 0, R).
R = 2.
?- verificar(1, 4, 5, 1, 1, R).
R = 4.
?- verificar(1, 4, 5, 1, 0, R).
R = “Contraseña descubierta”.
```

Su tarea es implementar esta nueva versión de *verificar*.

## 5. Red de tráfico

Al abrir la caja fuerte encuentra la última prueba adentro, un dibujo de un grafo incompleto con el siguiente enunciado “Se sospecha que un grupo de personas simbolizadas por estos círculos cometió un grave delito: tráfico de helados en pleno invierno. Se ha testificado que p7, p9, p13, p10 y p21 fueron vistos en la escena del crimen actuando sospechosamente, pero la *Pandilla en Línea recta* solo comete crímenes con todos sus integrantes siguiéndose en línea recta. Debes completar este dibujo sobre la escena del crimen, y descubrir quiénes son los miembros del grupo criminal.”. Junto al dibujo aparecen una serie de pistas que dicen: *sigue(p1, p2), sigue(p4, p5)*, y así sucesivamente, Herlock decide implementar estar pistas con Prolog, que ha usado todo este tiempo para resolver las pruebas.

Su tarea se enfoca en verificar quiénes son parte de una red principal de un grafo y quiénes no. Para esto, primero debe recrear el siguiente grafo conexo utilizando reglas del tipo *sigue(P1, P2)*:




Por ejemplo, la conexión entre *p1* y *p2*, *p2* y *p3* se ve de la siguiente forma:

```
sigue(p1, p2).
sigue(p2, p3).
```

… y así sucesivamente para todo el grafo

Nótese que bajo esta notación *p1* **sigue** a *p2*, no al revés, por lo que es distinto preguntar *sigue(p2, p1)*. Una vez realizado el grafo, debe armar un predicado que al recibir una de las personas determine si es parte de la ruta principal (denotada por la sección roja del grafo ilustrado), además de indicar el camino recorrido. Por ejemplo, *p11* no es parte de la ruta principal, a diferencia de *p8*, y así con las demás personas. A continuación, se enseña un ejemplo de la ejecución de las consultas, note que el arreglo que se enseña luego de cada consulta no es la respuesta, es solo un *print* conveniente:

```
?- principal(p7, R).

[p6,p5,p4,p3,p2,p1,p8]

R = "Es de la rama principal".

principal(p9, R).

[p1,p8,p7,p6,p5,p4,p3,p2]

R = "No es de la rama principal".
```

Como pista para realizar el algoritmo, note que si uno se devuelve desde cualquier punto del grafo, siempre llegará a la rama central, y si se continúa por ella terminará dando una vuelta completa, comenzando a repetir los nodos visitados. Considerar esto facilita la condición de salida del recorrido.

**NOTA: Solo** para este ejercicio se permite el uso de *append* en caso de requerirlo, y de *print* para mostrar por pantalla **únicamente la lista del recorrido**. Por último, se prohíbe resolver este problema de forma ad-hoc, es decir, enlistar mediante reglas cuáles son parte de la ruta principal significará puntaje 0 en la pregunta (por ejemplo: *ruta(p1) ruta(p2)…* para personas de la ruta principal).

## 6. Sobre la Entrega

- El código debe venir ordenado.
- Se deberá entregar un .tar.gz con los siguientes archivos:
    - p1.pl
    - p2.pl
    - p3.pl
    - p4.pl
    - README.txt
- Los códigos deben venir comentados, explicando clara y brevemente lo que realiza. Se deja libertad al formato del comentario.
- Los ayudantes correctores pueden realizar descuentos en caso de que el código se encuentre muy desordenado.
- Se debe trabajar de forma individual.
- La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea5LP_RolAlumno.tar.gz. Ejemplo: *Tarea5LP_202200000-0.tar.gz*
- El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la utilización de los programas.
- La entrega será vía aula y el plazo máximo de entrega es hasta el 15 de noviembre de 2023 a las 23:55 hora aula.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se pueden hacer consultas sobre la Tarea hasta 1 día antes de la fecha de entrega.

## 7. Calificación

### 7.1.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

### 7.1.2. Entrega mínima

La entrega mínima deberá tener la implementación del predicado que realiza lo pedido en **Descifrar el mensaje**, resolviendo correctamente los casos de prueba.

### 7.2. Entrega

- Implementar **La contraseña de la caja fuerte** (15pts)
- Implementar **Mejora para las pistas** (20pts)
- Implementar **Red de tráfico** (35pts)
    - Creación de reglas *sigue* (5 pts)
    - Identifica la pertenencia de cada persona a la ruta principal (30 pts).

Para todos los puntos, existe puntaje parcial según los casos de prueba que resuelve correctamente. Además, **no es necesario validar inputs** en ninguno de los problemas.

### 7.3. Descuentos

- Falta de comentarios (-5 pts c/u Max 20 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (entre -5 y -20 pts dependiendo de que tan desordenado)
- Día de atraso (-20 pts por día, -10 pts dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)